Build a connector

Use this guide to turn a bounded provider API into an AIP connector crate and a standalone connector-host binary. It is for Rust developers who own the provider mapping and can coordinate release, security, and deployment evidence.

SECTION	Connectors
TYPE	Guide
PROTOCOL	AIP 1.0
SOURCE	docs/guides/build-a-connector.md

The result of this procedure is a candidate connector artifact. Source code, a successful build, or a reachable health endpoint does not by itself establish admission, conformance, qualification, live-product compatibility, or production readiness.

This guide describes the implementation at source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. It covers AIP 1.0 and the frozen connector SDK boundary used by the connector fleet.

Use this procedure for a provider boundary

Build a connector when an external product has a stable operation or event surface that AIP needs to expose as capabilities. One connector type should own one implementation family. One connector instance should own one tenant-scoped provider account, workspace, application, endpoint set, or equivalent boundary.

Do not build a connector when the requirement is only to:

- translate AIP into MCP or A2A framing;
- add a trusted in-process module with no external product boundary;
- expose arbitrary caller-selected URLs, paths, headers, or credentials;
- rename an existing connector capability without changing its provider

contract.

Check the connector catalog first. Extend an existing connector when it already owns the provider boundary.

Prepare access and design inputs

Complete these prerequisites before changing the workspace:

- select the exact AIP source revision and Rust toolchain for the release;
- obtain the provider API specification and pin its version or revision;
- obtain a non-production provider account with the minimum required scopes;
- identify who may create credentials, webhooks, release signatures, evidence,

admission packages, and tenant bindings;

- define where durable runtime, replay, idempotency, and reconciliation state

will be stored;

- choose an immutable artifact format and digest;
- define a test topology that cannot affect production data.

Treat provider credentials, webhook secrets, signing seeds, database URLs, and private trust roots as deployment-owned secrets. They do not belong in source, the AIP manifest, capability input, registry catalog values, evidence payloads, logs, or documentation examples.

Before implementation, write a short boundary record:

Input	Decision to record	Why it matters
Provider identity	Product, API version, and upstream revision	Prevents an unbounded or moving API surface
Account boundary	Tenant plus provider account, workspace, app, or endpoints	Defines isolation and idempotency scope
Operation inventory	Exact reads, writes, deletes, messages, and long-running calls	Drives capabilities, risk, and approval
Authentication	Credential kind, required scopes, and rotation owner	Keeps caller input out of the secret boundary
Completion model	Synchronous, asynchronous, streaming, or provider job	Determines runtime and recovery behavior
Mutation behavior	Idempotency, retry, cancellation, reconciliation, compensation	Prevents duplicate or uncertain side effects
Ingress	Webhooks, signatures, timestamps, nonce, replay state, and limits	Defines the authenticated event boundary
Release evidence	Build, SBOM, provenance, conformance, and policy owners	Makes artifact admission reviewable

Stop if any mutation has unknown retry behavior and no conservative failure policy. A connector can publish fewer capabilities while the missing contract is resolved.

Create separate connector and host packages

Keep product mapping separate from deployment composition:

```
TEXT
crates/
├── aip-connector-provider-name/
│   ├── Cargo.toml
│   ├── src/
│   │   ├── lib.rs
│   │   └── operations.rs
│   └── tests/
│       ├── connector_contract.rs
│       └── frozen_conformance.rs
└── aip-host-provider-name/
    ├── Cargo.toml
    └── src/
        └── main.rs
```

Replace `provider-name` with the stable product identifier. Register both packages as workspace members and workspace dependencies.

The connector package normally depends on `aip-connector`, `aip-core`, `aip-discovery`, `aip-runtime`, `async-trait`, `serde_json`, and the bounded HTTP or sidecar client needed by the product. Add `aip-conformance` and schema tooling as development dependencies when the test driver uses them.

The host package depends on the product connector and the common `aip-connector-host` and `aip-connector-host-bootstrap` boundaries. Add routing or durable-store types only when product-owned ingress needs them. Do not embed the connector into the product-neutral `getaip-server` binary.

Use the connector crate for:

- provider configuration validation;

- capability and manifest construction;
- request and response mapping;
- typed failure classification;
- provider-specific idempotency, cancellation, and reconciliation;
- signed webhook validation before an event enters AIP.

Use the host crate for:

- parsing deployment configuration;
- reading owner-controlled secret files;
- constructing the product client;
- binding durable provider-specific stores;
- starting the common host lifecycle;
- mounting authenticated product ingress when required.

Publish an honest manifest

Implement `Connector` with a stable `id`, `discover`, `map_error`, and an appropriate `health` probe. The returned `Manifest` carries the manifest version, provider principal, capabilities, profiles, resources, channels, security, governance, limits, compatibility, and extensions.

Each callable capability needs a stable ID, input schema, optional output schema, and a contract that matches provider behavior. Record these properties before writing the provider call:

Contract area	What to declare
Side effects	Every applicable read, write, delete, send-message, financial, identity, medical, legal, external-network, or code-execution effect
Idempotency	Whether a key is required, its scope, duplicate behavior, and retention when known
Execution	Supported completion modes, cancellation, retry, and retry safety
Data	Sensitivity, retention, residency, and redaction behavior
Credentials	Required handle and scopes without secret material
Approval	Whether approval is required and what evidence is retained
Transactions	Implemented plan, commit, reconciliation, and compensation modes

Do not infer a safe contract from the HTTP method. A provider may implement a `POST` read, an asynchronous `DELETE`, or a write whose timeout leaves the outcome unknown.

Implement `CapabilityProviderConnector` when callers or tests need the capability list independently of the complete manifest. Return the same capability definitions from both surfaces.

Match runtime support to the contract

Implement `FrozenConnector` for the production execution boundary. Its `implementation_support` result is independent of the capability contract: the contract says what the capability promises, while the support map says what the current code implements.

Set each support flag only when the corresponding path exists:

- `invocation` for normal execution;
- `cancellation` when cancellation reaches the provider operation;

- `streaming` when incremental output reaches the runtime publisher;
- `retry` when retry classification and downstream idempotency are enforced;
- `transaction` for plan and commit;
- `reconciliation` for unknown provider outcomes;
- `compensation` for a separately governed compensation action;
- `approval` for approval evidence and resume behavior;
- `credentials` for deployment-resolved credential handles.

The discovery layer rejects claims such as streaming, cancellation, retry, or transaction support when the implementation map does not support them. It also rejects a callable capability without an implementation claim when that admission policy is enabled.

Every `FrozenConnector` operation receives `ActionExecutionContext`. Authorize and scope the provider request from its authenticated actor, verified tenant, credential handle, deadline, cancellation token, idempotency reservation, approval evidence, transaction state, trace context, and redaction policy. Use the supplied checkpoint and stream publishers for their declared purposes. Do not reconstruct authority from action input or metadata.

The host validates the signed gateway and the pinned route before it invokes the connector. That host-level route check is not caller-controlled connector metadata.

The frozen adapter rejects context-free execution. Keep that fail-closed behavior; do not add a second path that calls the provider without trusted context.

Operations that you do not implement already return `connector.operation_unsupported`. Override only the operations that the manifest and support map advertise.

Map provider failures conservatively

Return `ConnectorFailure` from typed operations. Preserve a stable namespaced code, a redacted message, AIP error category, retry decision, optional retry delay, provider request ID, durable provider operation reference, remote status, unknown-outcome flag, redacted details, source component, and connector operation.

Use the point of failure to decide retry and reconciliation:

Observation	Safe connector decision
Input rejected before dispatch	Permanent failure; no provider side effect
Authentication or scope rejected	Non-retryable until credentials or policy change
Provider rate limit on a retry-safe operation	Temporary and retryable with the provider delay when available
Read failed before a response	Retry only when the capability contract permits it
Mutation timed out after dispatch	Mark the outcome uncertain; do not report a safe blind retry
Provider returned an operation ID	Retain it for status checks, cancellation, or reconciliation
Cancellation won locally	Do not claim provider cancellation unless the remote endpoint confirmed it
Response exceeded the configured bound	Fail closed and retain only redacted diagnostic data

Never place provider payloads, credentials, authorization headers, signing material, or unbounded response bodies in `message` or `redacted_details`.

Protect secrets and provider destinations

Load credentials at the host boundary and wrap in `ConnectorSecret`. `ConnectorSecret` is non-serializable, redacts its debug output, compares in constant time, and zeroizes owned bytes on drop. Expose its bytes only while constructing the downstream request.

Validate provider destinations before accepting credentials:

- require HTTPS outside an explicit trusted development boundary;
- reject embedded usernames, passwords, fragments, and unexpected base paths;
- construct paths from admitted operations instead of caller-supplied URLs;
- allow only documented query keys, headers, redirects, and response types;
- apply request, response, timeout, and concurrency bounds;
- keep account or workspace identity in deployment configuration.

Use an opaque credential handle or secret-provider reference in registry and route state. A credential revision may be pinned to a route so an in-flight action cannot silently switch credentials during rotation.

Add ingress only when the provider needs it

Mount product routes with the common host only for authenticated provider ingress. Verify the signature over the exact raw body before JSON normalization. Validate the provider timestamp and account identity, reject replayed nonces or event IDs through durable state, and bound the body before parsing it.

Persist the accepted AIP event before acknowledging delivery when the provider retry contract requires durability. Publish to the central event endpoint through the host outbox so a transient central failure does not require a second provider delivery.

Ingress routes remain in the product host. They do not expand the central `getaip-server` router or bypass the connector's tenant boundary.

Compose the standalone host

Flatten `ConnectorHostBootstrapArgs` into the product host CLI. Product arguments add the provider origin, account identity, credential-file paths, enabled-operation configuration, and product limits.

The common bootstrap requires:

- the public native endpoint and narrow control-plane endpoint;
- admitted connector type, version, instance, and replica IDs;
- tenant and membership identity;
- gateway and control-plane verification DIDs;
- the exact immutable artifact digest;
- a durable PostgreSQL URL file and host signing-seed file;
- a non-secret secret-provider reference;
- topology, capacity, lease, heartbeat, drain, and health bounds.

The recurring host sequence is:

1. read and validate product configuration and secrets;
2. construct the connector;
3. call `PreparedConnectorHost::prepare`;
4. attach durable product stores or authenticated ingress;
5. call `serve`, `serve_with_router`, or `serve_with_router_factory`;
6. resolve shutdown through `shutdown_signal`.

`prepare` validates deployment identity, discovers the manifest, configures signing and trust, opens durable storage, and constructs the control-plane client. Serving then recovers durable runtime state, registers the replica, renews its lease, exposes the common HTTP surface, drains, and marks the replica offline.

The common surface is:

Route	Meaning
`GET /health`	Process and protocol identity only
`GET /ready`	Lease, drain, durable storage, and connector readiness
`GET /metrics`	Bounded connector-host metrics
`GET /aip/v1/manifest`	Exact running manifest
`POST /aip/v1/messages`	Signed, route-pinned native AIP execution

Do not use `/health` as a traffic gate. A host is ready only when `/ready` returns success and the registry sees an eligible lease.

Prepare immutable admission

Build the host as an immutable artifact and record its digest. Generate the manifest and implementation support map from the same source and configuration class. A production admission package binds:

- connector type and immutable version;
- manifest and canonical manifest digest;
- artifact digest and SDK version requirement;
- implementation support for every callable capability;
- admission policies, tenant-owned instances, pre-provisioned replicas, and

tenant capability bindings;

- seven mandatory evidence families.

The evidence families are OCI signature, SBOM, provenance, conformance, vulnerability policy, license policy, and revocation observation. Each evidence statement binds the artifact digest, manifest digest, document digest, signer, outcome, issue time, and expiration. Deployment trust policy supplies separate roots for the package and each evidence role.

Use the short-lived operator workflow after the release system has produced and signed the package:

```
SH
getaip connector registry plan \
  --package signed-admission.json \
  --trust-policy admission-trust-policy.json

getaip connector registry apply \
  --package signed-admission.json \
  --trust-policy admission-trust-policy.json \
  --database-url-file registry-database-url
```

`plan` verifies signatures, evidence, time bounds, digests, SDK compatibility, manifest invariants, and registry relationships without writing. `apply` uses a durable journal and can resume the same package revision and digest after interruption. A changed digest under the same identity is a conflict, not an update.

Do not give the long-running connector host registry-administrator credentials.

Test the provider boundary

Use layered tests so each result has a clear meaning:

1. Unit-test identifiers, schemas, path construction, header construction,

redaction, response bounds, and failure mapping.

2. Contract-test requests against a controlled provider stub. Include

idempotency collisions, reordered inputs, rate limits, timeouts, malformed responses, and ambiguous mutation outcomes.

3. Test webhook signatures, timestamps, replay fencing, account matching,

durable append, and central publication when ingress exists.

4. Implement a `ConnectorConformanceDriver` and exercise every scenario

implied by the manifest and implementation support.

5. Test host recovery, registration, lease loss, readiness, drain, offline

transition, credential rotation, and artifact mismatch.

6. Run an isolated live-provider matrix only with an authorized test account and

retained evidence.

The frozen conformance model contains twelve scenario families. They cover identity and credentials, schema enforcement, idempotency and duplicates, retry and exhaustion, cancellation races, streaming and backpressure, errors and uncertain outcomes, approval lifecycle, transaction lifecycle, audit and redaction, webhook security, and restart and reconnect.

A scenario applies according to the manifest and implementation support. An absent feature may make a scenario not applicable; it must not be reported as a passed implementation.

A source test demonstrates implementation behavior at that revision. Call a connector qualified only when an exact artifact, topology, procedure, timestamp, and retained result satisfy the declared qualification scope.

Roll out with a bounded first binding

Use this order for the first deployment:

1. Prepare a signed package with one non-production tenant binding and a

conservative admission policy, then verify it with `plan`.

2. Apply that exact signed admission package.
3. Deploy one replica with the exact admitted artifact digest and pre-provisioned

identity.

4. Wait for durable recovery, successful registration, a valid lease, and

`/ready`.

5. Compare the running manifest with the admitted manifest digest.
6. Discover the bound capability through the product-neutral gateway.
7. Invoke a read-only or otherwise non-destructive qualification capability.
8. Observe errors, lease state, capacity, latency, and retained audit evidence.
9. Expand bindings or replicas only after the bounded result is accepted.

Use `getaip connector test` for a basic host manifest check:

```
SH
HOST_URL=https://connector.example.test
```

```
CAPABILITY_ID=cap:twenty:metadata.list

getaip connector test "$HOST_URL" --capability "$CAPABILITY_ID"
```

The command confirms that the deployed endpoint advertises the requested capability. It does not prove tenant routing, provider execution, or qualification. This example deliberately omits `--invoke-capability`: a standalone host accepts signed actions only from its configured central gateway. Perform action qualification through that authorized route with an input and side-effect boundary approved for the test environment.

Verify the completed connector

Before requesting catalog publication, confirm all of the following:

- connector and host packages are separate and registered in the workspace;
- the provider boundary and upstream revision are fixed;
- manifest schemas and contracts match provider behavior;
- implementation support matches every advertised feature;
- typed execution uses trusted context and fails closed without it;
- failure, retry, cancellation, and uncertain-outcome behavior is tested;
- secrets stay in the host boundary and diagnostics remain redacted;
- ingress is authenticated, replay-fenced, bounded, and durable when present;
- the evidence and package reference the same artifact and manifest digests;
- registry identities and the running replica match the admitted IDs;
- the declared conformance and qualification scopes have retained results;
- rollback has been rehearsed without discarding durable reconciliation state.

Record source revision, provider revision, artifact digest, manifest digest, admission package ID and revision, tenant binding revision, test topology, timestamps, and evidence locations.

Decide failures without widening risk

Failure	Decision
Manifest admission fails	Correct the manifest or implementation support; do not weaken policy to publish it
Evidence is missing, stale, or mismatched	Rebuild the affected evidence for the exact artifact and manifest
Host registers with a different ID or digest	Stop the replica and correct deployment identity
`/health` succeeds but `/ready` fails	Inspect lease, drain, storage, and connector readiness before routing
Provider authentication fails	Disable the binding or drain the replica before rotating credentials
Mutation outcome is unknown	Preserve state and reconcile; do not send an unbounded retry
Lease renewal becomes ambiguous	Let the host replay its fenced request; do not create an untracked replica
Live-provider behavior differs from the contract	Disable the affected binding and reopen implementation review

Roll back without losing evidence

Stop new assignments by disabling the affected tenant binding or revoking the applied package revision. Drain the host before termination so assigned actions can finish within the configured deadline and the registry can record the offline transition.

Retain the connector runtime database, provider operation references, idempotency records, admission journal, signed package, evidence, logs, and metrics needed to resolve uncertain outcomes. Do not replace the artifact under an existing immutable version ID. Admit a corrected version and move bindings through a separately reviewed rollout.

Related documentation

- Capabilities
- Profiles and connectors
- Identity and trust
- Trusted local modules